



art news

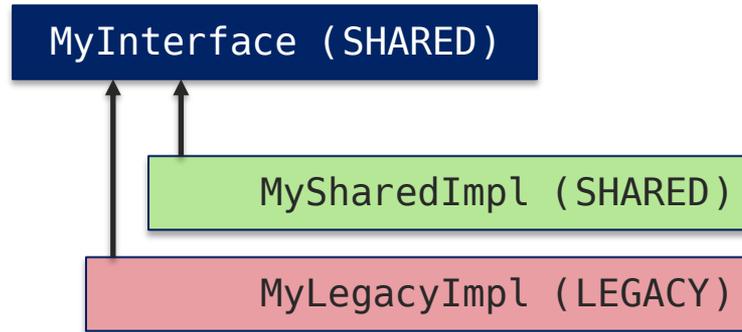
3 March 2020

art stakeholders meeting



art 3.05 (release sometime today)

- Same product stack as *art 3.04*
- Still supports SLF6 and macOS Mojave (SIP disabled)
- Minor bug fixes
- (Nearly) decouples service interface scope from implementation scope
 - LEGACY service interfaces must have LEGACY implementations
 - SHARED service interfaces may have either SHARED or LEGACY implementations
- One implementation's thread-safety has no bearing on another's



Potential *art* CVMFS area

- Until now, we have not provided a dedicated CVMFS area for *art* packages (and its dependencies)
- This has resulted in multiple installations of the same *art* versions (and dependencies) across experiment CVMFS areas
- If the *art/critic* suites and its dependencies were available from a dedicated CVMFS area, would the experiments see that as a benefit?

trigger_paths

- There are two reasons why you may wish to specify the `trigger_paths` parameter in your *art* job configuration:
 - You wish to enable execution for a subset of the configured trigger paths
 - You wish to specify the trigger bits

trigger_paths

- There are two reasons why you may wish to specify the `trigger_paths` parameter in your *art* job configuration:
 - You wish to enable execution for a subset of the configured trigger paths
 - You wish to specify the trigger bits

```
physics: {  
  ...  
  a: [m1, m2] # index 0  
  b: [m1, m3] # index 1  
  c: [m4, m5] # index 2  
}
```

trigger_paths

- There are two reasons why you may wish to specify the `trigger_paths` parameter in your *art* job configuration:
 - You wish to enable execution for a subset of the configured trigger paths
 - You wish to specify the trigger bits

```
physics: {  
  ...  
  a: [m1, m2] # index 0  
  b: [m1, m3] # index 1  
  c: [m4, m5] # index 2  
}
```

```
physics: {  
  ...  
  a: [m1, m2] # index 0  
  b: [m1, m3] # disabled  
  c: [m4, m5] # index 1  
  trigger_paths: [a, c]  
}
```

trigger_paths

- There are two reasons why you may wish to specify the `trigger_paths` parameter in your *art* job configuration:
 - You wish to enable execution for a subset of the configured trigger paths
 - You wish to specify the trigger bits

```
physics: {  
  ...  
  a: [m1, m2] # index 0  
  b: [m1, m3] # index 1  
  c: [m4, m5] # index 2  
}
```

```
physics: {  
  ...  
  a: [m1, m2] # index 0  
  b: [m1, m3] # disabled  
  c: [m4, m5] # index 1  
  trigger_paths: [a, c]  
}
```

```
physics: {  
  ... # Should be  
  a: [m1, m2] # index 1  
  b: [m1, m3] # disabled  
  c: [m4, m5] # index 0  
  trigger_paths: [c, a]  
}
```

trigger_paths

- There are two reasons why you may wish to specify the `trigger_paths` parameter in your *art* job configuration:
 - You wish to enable execution for a subset of the configured trigger paths
 - You wish to specify the trigger bits

```
physics: {  
  ...  
  a: [m1, m2] # index 0  
  b: [m1, m3] # index 1  
  c: [m4, m5] # index 2  
}
```

```
physics: {  
  ...  
  a: [m1, m2] # index 0  
  b: [m1, m3] # disabled  
  c: [m4, m5] # index 1  
  trigger_paths: [a, c]  
}
```

```
physics: {  
  ... # Actually is  
  a: [m1, m2] # index 0 ☹️  
  b: [m1, m3] # disabled  
  c: [m4, m5] # index 1 ☹️  
  trigger_paths: [c, a]  
}
```

trigger_paths

- There are two reasons why you may wish to specify the `trigger_paths` parameter in your *art* job configuration:
 - You wish to enable execution for a subset of the configured trigger paths
 - You wish to specify the trigger bits

```
physics: {  
  ...  
  a: [m1, m2] # index 0  
  b: [m1, m3] # index 1  
  c: [m4, m5] # index 2  
}
```

```
physics: {  
  ...  
  a: [m1, m2] # index 0  
  b: [m1, m3] # disabled  
  c: [m4, m5] # index 1  
  trigger_paths: [a, c]  
}
```

```
physics: {  
  ... # Actually is  
  a: [m1, m2] # index 0 ☹️  
  b: [m1, m3] # disabled  
  c: [m4, m5] # index 1 ☹️  
  trigger_paths: [c, a]  
}
```

- We would like to fix the framework to reflect expected behavior.
- This *could* break current workflows. Is there a concern about this?

C++ features

- C++20 has been finalized.
 - Draft International Standard out for ISO approval
- The 4 main core features:
 - Concepts
 - Coroutines
 - Modules
 - Ranges
- STL additions:
 - format library
 - `std::span`
 - `std::source_location`
 - `constexpr std::string/std::vector`
- No specific plan as to when C++20-enabled builds of *art* would be available.
- Current compiler status for C++20 can be found:
 - GCC: <https://gcc.gnu.org/projects/cxx-status.html#cxx2a>
 - Clang: https://clang.llvm.org/cxx_status.html#cxx20

